Lab: **AI & ML Lab (23A31403)**

College: Andhra

Engineering College

9. Apply KNN algorithm for classification and regression

9a-Classifier).KNeighborsClassifier
9a-Regressor).KNeighborsRegressor
9b – Classifier).RadiusNeighborsClassifier
9b-Regressor).RadiusNeighborsRegressor

9a-Classifier).KNeighborsClassifier

Mathworks Here Metric May Changes Based on Problem Statement

2.2.1 Minkowski Distance

The Minkowski metric is a commonly used family of distance metrics which can be expressed as

$$d^r(p,q) = \left(\sum_{k=1}^{L}(|p_k - q_k|^r)\right)^{\frac{1}{r}},$$

where r is a parameter that determines the type of metric being used and p and q are l-dimensional vectors. Some variations based on selecting the value of r are:

- L_∞ *norm:* Here, $r = \infty$ and $d(p,q) = maximum_k(|p(k) - q(k)|)$, $k \in \{1,\ldots,L\}$.
- L_2 *norm:* In this case, $r = 2$ and $d(p,q) = (\sum_{k=1}^{L}(|p(k) - q(k)|^2))^{\frac{1}{2}}$ is the Euclidean distance; this is the most popular variation.
- L_1 *norm:* In this case, $r = 1$ and $d(p,q) = (\sum_{k=1}^{L}(|p(k) - q(k)|))$ is the city-block distance.
- *Fractional norm:* It is possible that r is a fraction. In such a case, the resulting distance is called fractional norm. It is not a metric as it violates the triangle inequality.

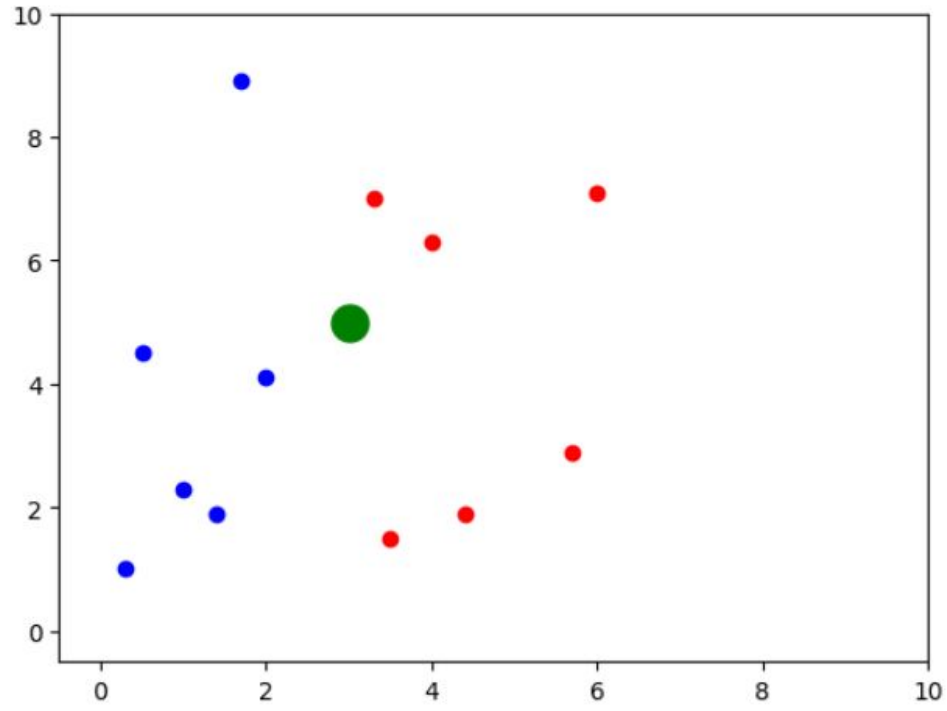
The importance of different norms will be examined while explaining the nearest neighbor classifiers. It is important to ensure that all features used in the distance measure have the same range of values, as attributes with larger ranges may gain undue advantage. **Normalisation** of feature values can help to ensure that they are in the same range.

Program :

```
import numpy as np
from matplotlib import pyplot as plt
from sklearn.neighbors import KNeighborsClassifier

xBlue = np.array([0.3,0.5,1,1.4,1.7,2])
yBlue = np.array([1,4.5,2.3,1.9,8.9,4.1])

xRed = np.array([3.3,3.5,4,4.4,5.7,6])
yRed = np.array([7,1.5,6.3,1.9,2.9,7.1])

X =
np.array([[0.3,1],[0.5,4.5],[1,2.3],[1.4,1.9],[1.7,8.9],[2,4.1],[3.3,7],[3.5,1.5],[4,6.3],[4.4,1.9],[5.7,2.9],[6,7.
1]])
y = np.array([0,0,0,0,0,0,1,1,1,1,1,1]) # 0: blue class, 1: red class

plt.plot(xBlue, yBlue, 'ro', color = 'blue')
plt.plot(xRed, yRed, 'ro', color='red')
plt.plot(3,5,'ro',color='green', markersize=15)
plt.axis([-0.5,10,-0.5,10])

classifier = KNeighborsClassifier(n_neighbors=3) # this is the k value
classifier.fit(X,y)

pred = classifier.predict(np.array([[5,5]]))
print(pred)

plt.show()
```

Output:

```
  plt.plot(3,5,'ro',color='green', markersize=15)
[1]
```



9a-Regressor).KNeighborsRegressor
Mathworks

2.4 KNN REGRESSION

It is possible to use KNN for regression also. So, in this case we are given a set $\mathcal{X}$ of n labelled examples, where

$$\mathcal{X} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$$

Here x_i, $i = 1, 2, \ldots, n$ is a data vector and y_i is a scalar. It is possible for y_i also to be a vector in some applications. However, we restrict our attention to scalar y_is. The regression model needs to use $\mathcal{X}$ to find the value of y for a new vector x. In the case of regression based on KNN, we perform the following:

1. Find the k nearest neighbors of x from n data vectors. Let them be $x^1, x^2, \ldots, x^k$.
2. Consider the y values associated with these x's. Let them be $y^1, y^2, \ldots, y^k$.

3. Take the average of these y's and declare this average value to be the predicted value of y associated with x. So, the predicted value of y, call it $\hat{y}$ is,

$$\hat{y} = \frac{1}{k}(y^1 + y^2 + \cdots + y^k)$$

We will illustrate it using the following example.

EXAMPLE 15: Consider the data shown in Table 2.10.

TABLE 2.10 Example data for KNN regression

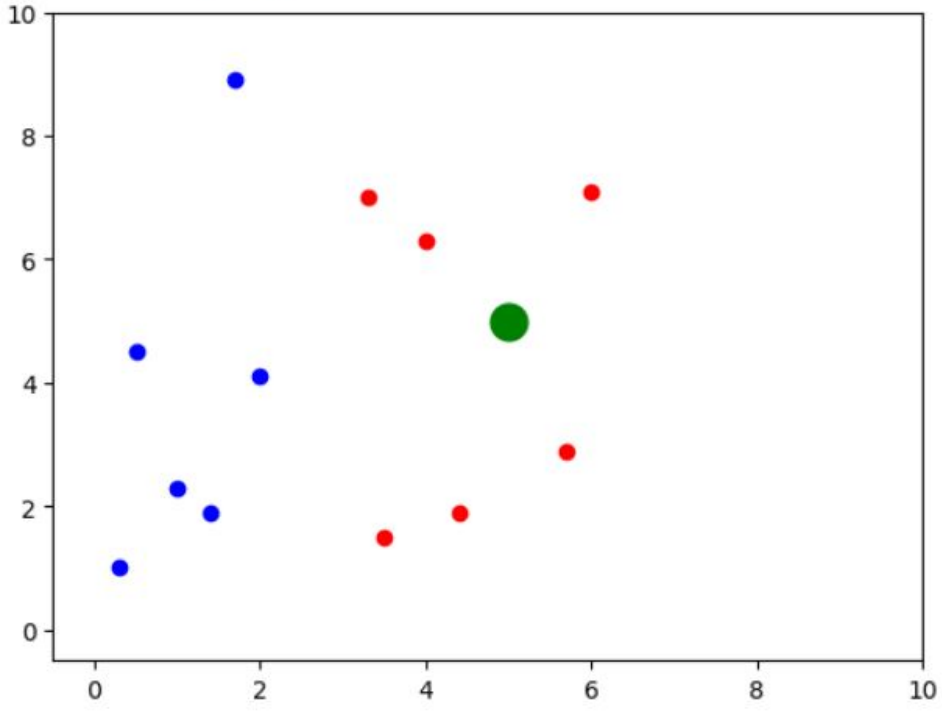
Number (i)	Pattern (x_i)	Target (y_i)
1	(0.2,0.4)	8
2	(0.4,0.2)	8
3	(0.6,0.4)	12
4	(0.8,0.6)	16

Program :

```
import numpy as np
from matplotlib import pyplot as plt
from sklearn.neighbors import KNeighborsRegressor

# ----------------------------
# DATA (same points)
# ----------------------------
xBlue = np.array([0.3,0.5,1,1.4,1.7,2])
yBlue = np.array([1,4.5,2.3,1.9,8.9,4.1])

xRed = np.array([3.3,3.5,4,4.4,5.7,6])
yRed = np.array([7,1.5,6.3,1.9,2.9,7.1])

# Feature matrix
X = np.array([
   [0.3,1],[0.5,4.5],[1,2.3],[1.4,1.9],[1.7,8.9],[2,4.1],
   [3.3,7],[3.5,1.5],[4,6.3],[4.4,1.9],[5.7,2.9],[6,7.1]
])

# ----------------------------
# TARGET VALUES (continuous)
# ----------------------------
# Example: any numeric value (e.g., temperature, price, score)
y = np.array([10,12,11,13,15,14,20,18,22,19,21,23])

# ----------------------------
# PLOTTING
# ----------------------------
plt.plot(xBlue, yBlue, 'ro', color='blue')
plt.plot(xRed, yRed, 'ro', color='red')

# Point to predict
plt.plot(5,5,'ro', color='green', markersize=15)
plt.axis([-0.5,10,-0.5,10])

# ----------------------------
# KNN REGRESSOR
# ----------------------------
regressor = KNeighborsRegressor(n_neighbors=3)
regressor.fit(X, y)

# Prediction
pred = regressor.predict(np.array([[5,5]]))
print("Predicted value:", pred)

plt.show()
```

Output :

```
Predicted value: [22.]
```



9b – Classifier).RadiusNeighborsClassifier

Mathworks

2.3.4 Radius Distance Nearest Neighbor Algorithm

This algorithm is an alternative to the KNN algorithm that considers all the neighbors within a specified distance r of the point of interest. This algorithm can be described as follows:

1. Given a point T, identify the subset of data points that fall within the radius r centred at T, denoted by
$$Br(T) = \{x_i \in \mathcal{X} \; s.t. \; \|T - Xi\| \leq r\}$$
2. If $Br(T)$ is empty, output the majority class of the entire data set.
3. If $Br(T)$ is not empty, output the majority class of the data points within $Br(T)$.

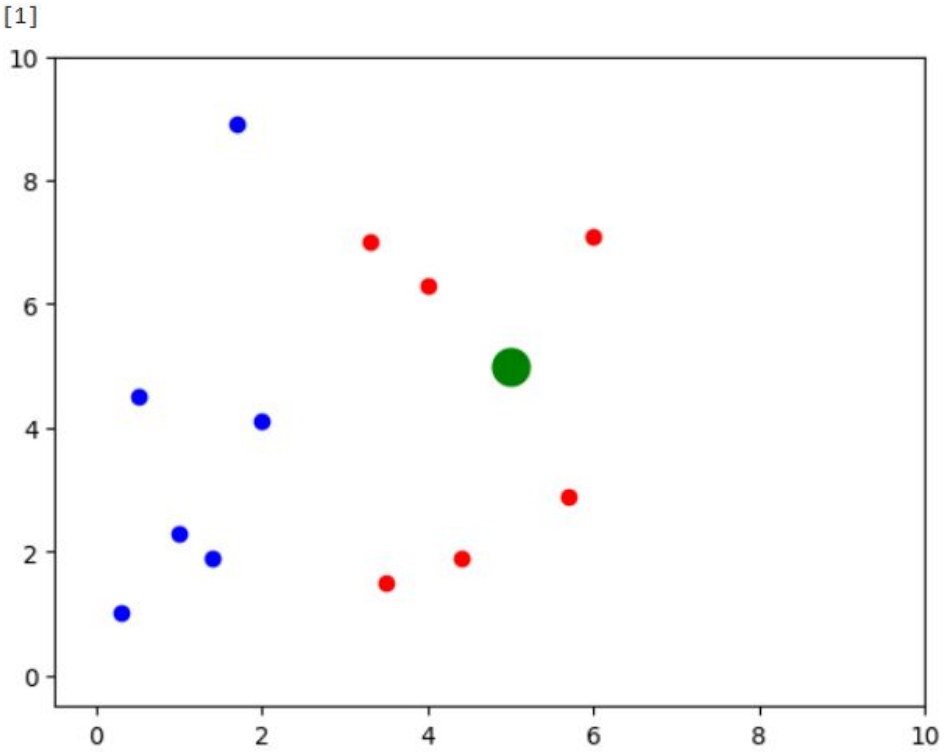
This algorithm is useful for identifying outliers, as any pattern that does not have similarity with the patterns within the chosen radius can be identified as an outlier. The choice of the value of radius r is critical as it can affect the performance of the algorithm.

Program :

```
import numpy as np
from matplotlib import pyplot as plt
from sklearn.neighbors import RadiusNeighborsClassifier

xBlue = np.array([0.3,0.5,1,1.4,1.7,2])
yBlue = np.array([1,4.5,2.3,1.9,8.9,4.1])

xRed = np.array([3.3,3.5,4,4.4,5.7,6])
yRed = np.array([7,1.5,6.3,1.9,2.9,7.1])

X = np.array([
   [0.3,1],[0.5,4.5],[1,2.3],[1.4,1.9],[1.7,8.9],[2,4.1],
   [3.3,7],[3.5,1.5],[4,6.3],[4.4,1.9],[5.7,2.9],[6,7.1]
])

y = np.array([0,0,0,0,0,0,1,1,1,1,1,1])  # 0: blue class, 1: red class

plt.plot(xBlue, yBlue, 'ro', color='blue')
plt.plot(xRed, yRed, 'ro', color='red')
plt.plot(5,5,'ro', color='green', markersize=15)
plt.axis([-0.5,10,-0.5,10])

# □ Radius Neighbors Classifier
classifier = RadiusNeighborsClassifier(radius=2.5)
classifier.fit(X, y)

pred = classifier.predict(np.array([[5,5]]))
print(pred)
```

plt.show()

Output:



9b-Regressor).RadiusNeighborsRegressor

Program :

```
import numpy as np
from matplotlib import pyplot as plt
from sklearn.neighbors import RadiusNeighborsRegressor

xBlue = np.array([0.3,0.5,1,1.4,1.7,2])
yBlue = np.array([1,4.5,2.3,1.9,8.9,4.1])

xRed = np.array([3.3,3.5,4,4.4,5.7,6])
yRed = np.array([7,1.5,6.3,1.9,2.9,7.1])

X = np.array([
   [0.3,1],[0.5,4.5],[1,2.3],[1.4,1.9],[1.7,8.9],[2,4.1],
   [3.3,7],[3.5,1.5],[4,6.3],[4.4,1.9],[5.7,2.9],[6,7.1]
])

# □ Continuous target values (for regression)
y = np.array([10,12,11,13,15,14,20,18,22,19,21,23])

plt.plot(xBlue, yBlue, 'ro', color='blue')
```

```
plt.plot(xRed, yRed, 'ro', color='red')
plt.plot(5,5,'ro', color='green', markersize=15)
plt.axis([-0.5,10,-0.5,10])

# □ Radius Neighbors Regressor
regressor = RadiusNeighborsRegressor(radius=2.5)
regressor.fit(X, y)

pred = regressor.predict(np.array([[5,5]]))
print(pred)

plt.show()
```

Output:

```
[22.]
```